Exercise 2 - Concat vs Append

This experiment is based on five runs of the program and the results for each run are stored in the tables below. The last table contains the average values for the experiment.

First of all I saved the current time in a long-variable called “before” using nanoTime(). Then I have a while-statement saying that if the difference of the current time and the time before is less than 1 second, continue to concat/append strings. I also have a variable that increase +1 each time a concat/append is made. When one second has passed a result should be ready to be printed out with the amount of concats/appends and the length of the string. This method is the same for each approach.

Run 1	Times	String length
Short concat	30101	30101
Short append	76080	76080
Long concat	3899	311920
Long append	8657	692560

Run 2	Times	String length
Short concat	30389	30389
Short append	71701	71701
Long concat	3426	274080
Long append	8103	648240

Run 3	Times	String length
Short concat	30726	30726
Short append	68256	68256
Long concat	3777	302160
Long append	7478	598240

Run 4	Times	String length
Short concat	31120	31120
Short append	72160	72160
Long concat	3835	306800
Long append	8547	683760

Run 5	Times	String length
Short concat	30335	30335
Short append	74359	74359
Long concat	3881	310480
Long append	8478	678240

Average	Times	String length
Short concat	30534	30534
Short append	72511	72511
Long concat	3764	301088
Long append	8253	660208

Why is StringBuilder faster than String concatenation?

As I understand String concatenation is much slower because it creates a new String each time you concat, so it has to copy the old String and add the new String to it. So in theory that would mean that it also uses more memory?

A StringBuilder works differently as it takes each char of the String and adds it to it's internal array, which is a much faster approach.

Exercise 3 - Sorting Algorithms

This experiment is based on five runs of the program and the results for each run are stored in the tables below. The last table contains the average values for the experiment.

I start with an array with the size of 1000. Then I generate the same amount of random integers within the interval 0 - 100000. Now the clock starts and the insertionSort() is called to sort the array. After the array is sorted I check that the time is still under one second so that **lessThanOneSecond = true**. The array are now getting new values in order to do the same process again. After one second has passed, the amount of sorted integers will be returned.

The insertion sort for strings is done in a similar way, but the random generator for integers is now replaced by a method getRandomString() that randomizes a string with ten letters.

Run 1	Sorted ints/strings
Integers	3932000
Strings	558000

Run 2	Sorted ints/strings
Integers	4284000
Strings	618000

Run 3	Sorted ints/strings
Integers	4151000
Strings	531000

Run 4	Sorted ints/strings
Integers	4370000
Strings	592000

Run 5	Sorted ints/strings
Integers	4220000
Strings	504000

Average	Sorted ints/strings
Integers	4191400
Strings	560600

It’s hard to say if i’m satisfied with the results since you’re not aware of what a realistic result would be. Anyhow I managed to get pretty much the same values every time I ran the program.

ga222gb